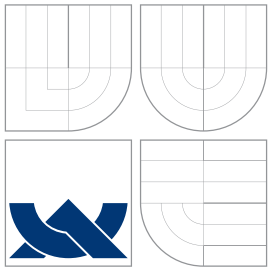
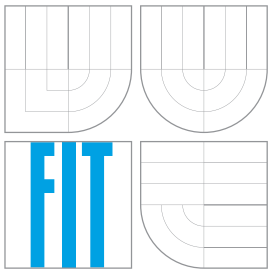


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

# MULTIPLAYER HRA PRO PLATFORMU ANDROID

MULTIPLAYER GAME FOR ANDROID PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ANDREJ MARTINÁK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PAVEL ŽÁK

BRNO 2013

# Abstrakt

Tato bakalářská práce se věnuje problematice vývoje real-time mobilních her pro více hráčů na platformě Android. První kapitoly se zabívají analýzou a možnostmi vývoje aplikací na této platformě. Následuje popis různých principů používaných při tvorbě her pro více hráčů. Součástí práce je také popis technologií, které byli analyzovány a poté použity při implementaci 2D real-time bojové hry a herního serveru. V práci je postupně popsán proces návrhu, implementace hry a testování celé aplikace.



# Abstract

This bachelor thesis deals with issues of real-time multiplayer mobile games development on Android platform. First chapters are devoted to the analysis of this platform and its possibilities of applications development. This is followed by the description of differenent approaches used for multiplayer game development. Part of the thesis describes different technologies, that were analyzed and then used for implementation of a 2D real-time fighting game and a server. The thesis contains the description of the game design, implementation and testing processes of the application.



# Klíčová slova

Více hráčů, real-time, hra, Android, Google, mobilné zařízení, mobilní aplikace, síť, AndEngine, SmartFox, SmartFoxServer, server

# Keywords

Multiplayer, real-time, game, Android, Google, mobile device, mobile application, network, AndEngine, SmartFox, SmartFoxServer, server

# Citace

Andrej Martinák: Multiplayer game for Android platform, bakalářská práce, Brno, FIT VUT v Brně, 2013

# Multiplayer game for Android platform

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Pavla Žáka

.....
Andrej Martinák
May 12, 2013

## Poděkování

Týmto by som chcel poďakovať Ing. Pavlovi Žákovi, že si vzal túto tému pod svoje vedenie, za jeho podnety a rady. Ďalej chcem poďakovať všetkým ľuďom, ktorí sa zúčastnili testovania aplikácie.

# Contents

# Chapter 1

# Introduction

Multiplayer gaming on mobile devices is relatively new. Mobile devices, in the past, were often used by managers and experts mostly for business and organizing purposes. The ownership of smartphones and tablets today, though, is quite common and it is still rising. This fact indicates, that these devices have opened multiplayer mobile gaming possibilities to a wide audience. The mobile devices used to lack network and computational power resources and the multiplayer gaming in the past was targeted mostly on PC and video game consoles. Mobile game were usually single-player or simple turn-based games, that could be played by multiple players on one device. However, today's mobile devices with multi-core processors and 4G networks (or Wi-Fi connection) provide enough power to run any game.

The topic of the thesis was an idea of my own. The goal of this thesis was to create a real-time multiplayer fighting game for Android, based on the client-server model. I chose to pick this topic, because of the lack of real-time multiplayer games on the Android market. The development of this game was focused mainly on the creation of stable multiplayer connection and making the game fun to play.

The first chapter of the thesis introduces the Android platform. Next chapter discusses the development on this platform in general. Multiplayer game development theory and its basics are discussed in the chapter number 4. After this chapter, the reader can find a list of today's gaming network frameworks, that I analyzed and also the game engine I used. The next two chapters are focused on the design, implementation and testing processes of the game development. In the last chapter, there is an evaluation of the whole thesis. It also discusses the possibilities of the next development.

# Chapter 2

# Android platform

This platform is on the market just shortly, but since the time of its creation it has earned hundreds of millions of users, fans and supporters. This chapter discusses and analyses the platform as it is. The reader will get to know, what Android is, everything necessary about its history and what versions this platform has. He will also find out, how Android architecture looks like.

## 2.1 Platform introduction

The Android from Google contains these three elements: operating system, middleware and key applications [11]. It is focused for mobile devices like smartphones or tablets. The platform is licensed as open source (Apache Software License, 2.0).

The operating system based on Linux kernel is optimized for different devices with different hardware. Therefore, it is not constricted with chipset, screen size or the screen resolution, that the devices uses.

The platform contains/supports the following features:

- Application framework
- Dalvik virtual machine – virtual machine optimized for mobile devices
- Integrated web browser – based on open-source engine WebKit
- Optimized graphics – uses its own 2d graphical libraries. 3D graphics is based on OpenGL ES 1.0
- SQLite – for data structure
- Media support – well known sound, video and picture formats(MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- GSM technology (depending on hardware)
- Bluetooth, EDGE, 3G, WiFi (depending on hardware)
- Camera, GPS, compas, accelerometer (depending on hardware)

### Android history

The platform was created by Android, Inc. – settled in Palo Alto, California, in the city habited by many significant companies (like Hewlett-Packard, VMware, Facebook). The company was founded in 2003. It was bought by Google in 2005 and so it became its subcompany. On November 5th, 2007 34 technological companies like Google, HTC, Motorola, Intel, Qualcomm, NVIDIA, T-Mobile, Samsung Electronics, etc. came together with an idea to create a congregation. And so Open Handset Alliance (OHA) was created. The goal of this consortium was to create an open standard for mobile devices. In the same day OHA revealed its first product – Android, an open mobile platform based on Linux kernel version 2.6. A week later OHA also released SDK for developers. The first Android-based commercial device was officially the phone HTC Dream (also known as T-Mobile G1), which was available since October 22nd, 2008.

### Platform versions

Until today 29 versions of Android has been released. Platform was originally focused on smartphones, later the focus moved to tablets as well (Honeycomb version). Particular versions are named by desserts.

The latest version (Jelly Bean) was announced on June 27, 2012 on Google I/O conference. This update came with the aim of improving the user interface and the performance.

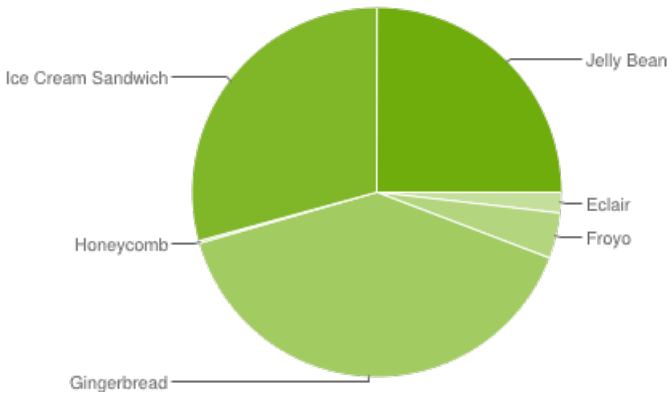



Figure 2.1: Statistic of usage of particular Android versions (April 2nd, 2013) [13]

## 2.2 Android architecture

The Architecture of Android can be divided into 5 main parts: Applications, Application framework, Libraries, Android Runtime and Linux kernel. Particular parts of the platform and their parts are shown on figure 2.2.

### Applications

Android is supplied with some essential applications like email client, SMS application, maps, browser, contacts, etc. Applications are mostly written in Java language.

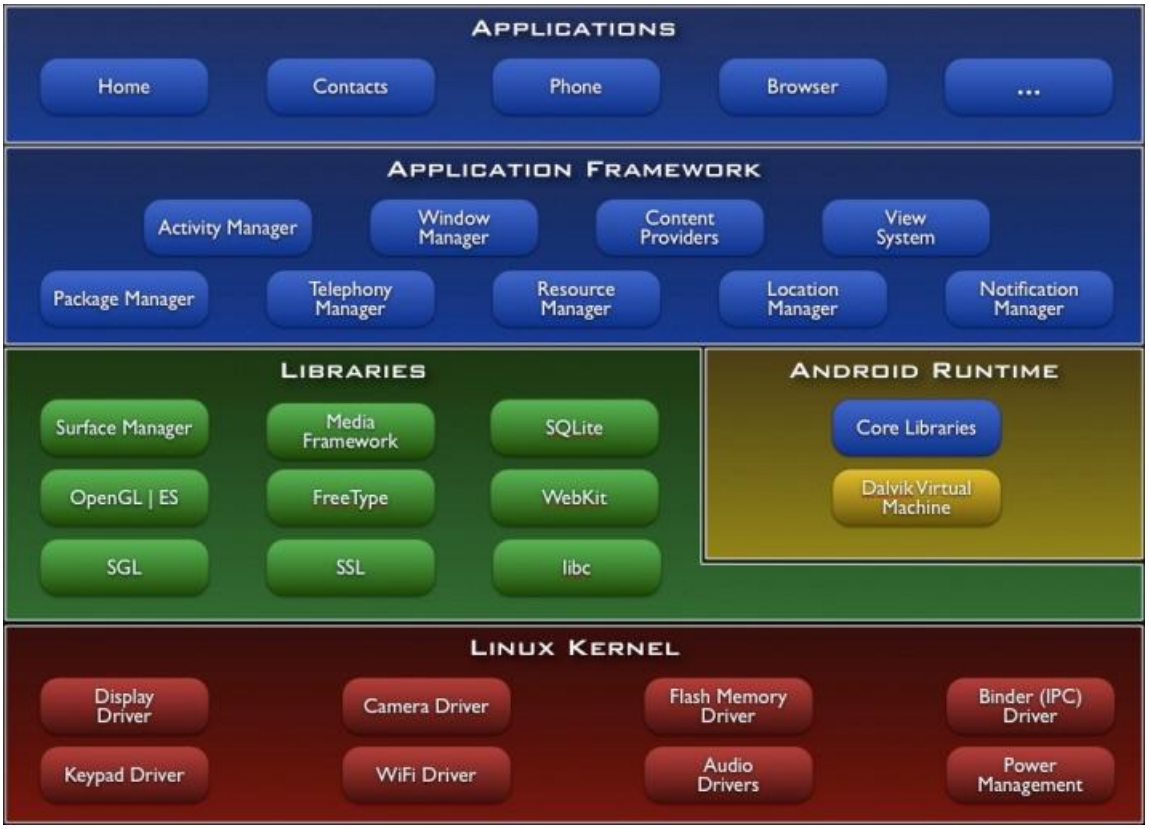



Figure 2.2: The architecture of Android platform

### Application framework

Being an open platform, Android allows developers to create rich and innovative applications. The programmer has access to all parts of the device - from hardware parts (e.g. diodes), notifications, background services to alarms, etc. The application framework allows the programmer to use built-in libraries and contains all necessary classes and interfaces needed to create an application. The architecture of applications was designed to simplify the usage of repeating components.

### Libraries

Android contains libraries written in C/C++. They are used by various components of the system. These functions are provided to the developer just by the application framework. Main libraries are:

- libc – standard C library modified for embedded devices, that are using Linux kernel
- Media libraries – libraries for image and audio/video playback and recording support
- SQLite – simplified relation database library
- FreeType – library for bitmap and vector font rendering
- 3D libraries – implemented on OpenGL ES 1.0 API base

### Android Runtime

This layer contains Dalvik Virtual Machine (DVM) – virtual machine optimized for mobile devices (energy saving, stamina). It also contains base Java programming language libraries.

Virtual machine Dalvik provides hardware independence of Android. It is a simulation of a processor and has its own instruction set (dex-code – Dalvik Executable). Applications are compiled with the compiler from the source code to DVM instruction code (these instructions are also called Java bytecode), which is launched on the DVM. Every application runs on its own process and has its own instance of the DVM.

### Kernel

The kernel is the lowest layer of the architecture, which forms an abstract layer between hardware and software. It is built on the Linux kernel. However, it does not contain native support for X-Window Server and does not contain all standard GNU libraries. Therefore, it is not possible to run on Android any application written in Java language. The kernel contains drivers needed for usage of all hardware components. It also contains services for security, memory management or process management. The Linux kernel was picked for its wide possibilities of portability.

# Chapter 3

# Android application development

Developing an application for a mobile device may seem really complicated. In history, it was necessary to adjust the application development to a certain device with defined hardware. Various companies have different solutions. For example, Apple simplifies the development by having just a small amount of devices – iPad and iPhone and various version of iOS operating system. However, the goal of Android development is creating a platform that can be accessed by all mobile devices, no matter what hardware the device has. In this chapter the reader will find out, what he needs to develop on Android and also all necessary things needed for developing applications for Android.

## 3.1 Android development challenges

Aiming the application development to the Android platform requires the knowledge of some base factors:

- Device displays are small and every device can have a different resolution with a different density. Because of that it is necessary to develop applications in a way that they can look appropriately on most devices.
- The speed of CPU and size of the memory are limited. It is not allowable, that the application uses the processor fully. Also it can not cause a system crash because of memory leaks. Applications must work reliably and can not delimit other applications and processes that are currently launched on the device.
- The application works on a phone. So its primary functions are calling and accepting calls. Therefore, it is not allowable, that these functions are anyhow limited by another running application (hanging applications, crush by a call). Memory leaks are again related with this.

## 3.2 Android SDK

Because Android is and open-source platform, it is possible to download the development tools *Android SDK (Software Development Kit)* of all Android versions from official websites for developers [12]. The SDK can be downloaded for Windows, Linux and Mac OS X as well.

The absolute essence for a developer is the *Android Development Tools* plugin for Eclipse IDE. This environment (recommended and used by Google developers) creates and easy and intuitive way of Android application development and debugging. After the installation it is just needed to set path to the SDK. This package contains tenths of tools. Let us just mention the most important:

### Android Virtual Device

A part of the SDK package is also an emulator of Android devices. There is a big amount of Android devices with various hardware, so it is possible to emulate any device and test applications on it.

### Android Debug Bridge

Using this tool developers can create a communication between the computer and device (emulated or physical – connected via USB). It is a *client-server* program and it can be in these states:

- client – runs on the computer, where application is developed. It can communicate with devices with various commands.
- server – runs on the computer as a process. It allows the communication between client and daemon.
- daemon – runs on the background of every emulated or physical device.

## 3.3 Basic application structure

Android applications consist of these components [14]:

- **Activity** – is the base element of user interface (UI). Activities can be understood as one logical unit (one dialog window in a desktop application. It fills the screen with content, that user can interact with (e.g. send SMS, accept a call, read an email, etc.). Applications consist mostly from more activities, which are linked together. Every application has the *main activity*, which can run another activity. The life cycle of an activity is figured on figure 3.1.
- **Content provider** – using *content provider* you can store and retrieve data and send them to applications. They offer an abstraction layer for any data stored on the device. They can be accesable by more applications. It is one of ways how to share data between them.
- **Service** – *activity* and *content provider* have short lifetime and can be stopped at any time. That is why *services* exist. They run on the background (if necessary) and are independent of other activities. They do not provide any user interface. *Services* are, for example, used by RSS readers for checking new news or music players – so music can play on the background.
- **Intent** – *activities* or *services* are activated by these system messages. They are used to signalize some events for applications - e.g. hardware state change (SD card inserted), incoming data (SMS) or events from other applications.

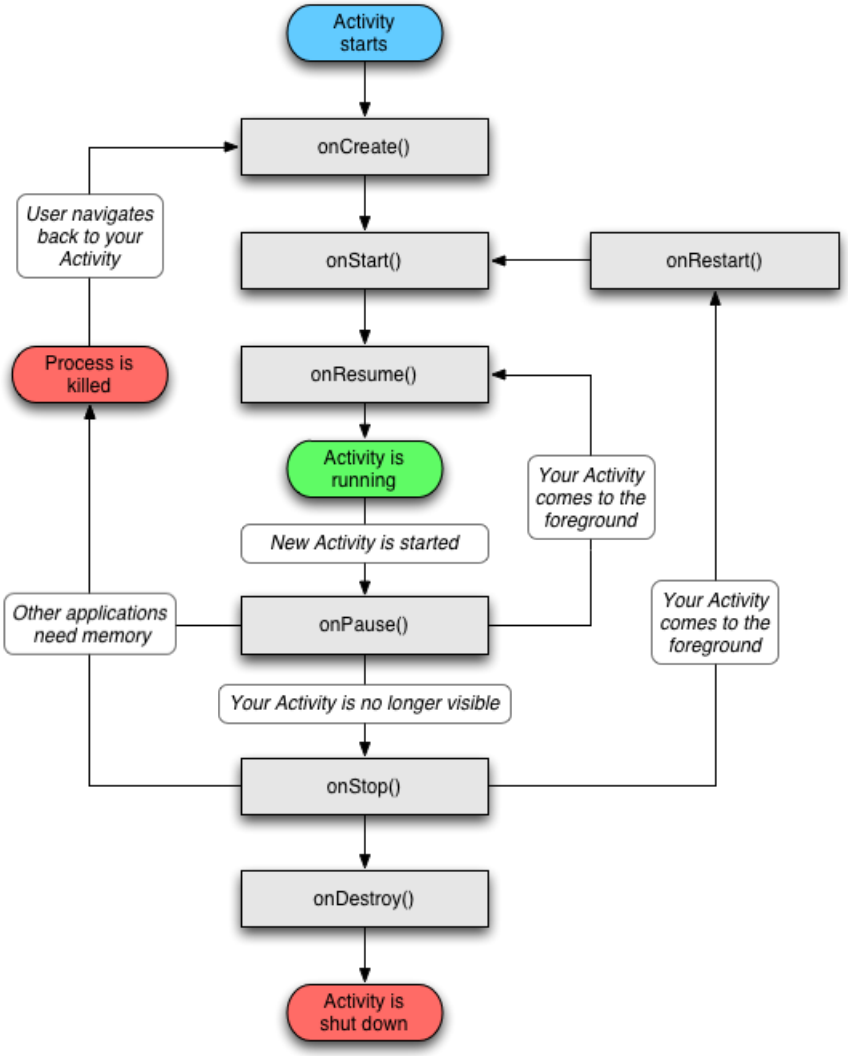



Figure 3.1: Activity life cycle [10]

- **Processes and threads** – they work just like on Linux. By launching an application (by default) a new process is created. Application and all its parts (*main thread*) are running in it. From this process it is possible to create new processes or threads.

## 3.4 Creating a user interface

As it was mentioned in chapter 2.2, applications are written in Java programming language. But compared to desktop computers, Android devices use touchscreens controlled by fingers. That is why the creation of a user interfaces requires different approach. It is done by using XML files, that support the possibility of structuring the hierarchy of objects - figure 3.2. These XML files with special tags can be edited in any notepad, or with a graphical editor, which is a part of the development kit for Eclipse.

The main element of the user interface XML files are *View* and *ViewGroups* objects, of

which other user interface objects are extended (such as buttons, lists, images etc.).

It is possible to save the composition of these objects to *Layouts*. By using layouts it is possible to create a user interface that can look properly on different screens (with different resolutions and density).

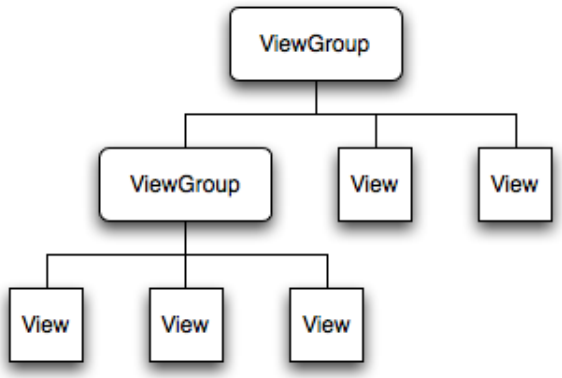



Figure 3.2: View and ViewGroup objects hierarchy

# Chapter 4

# Multiplayer game development

Multiplayer games are games played between multiple players in the same environment. Nowadays, genres like MMORPG [1], RTS [2] or FPS [3] provide communication between hundreds to thousands players at the same time. This simultaneous gaming experience offers players a possibility to compete against each other or cooperate together in order to win the game and have fun. However, creating the feeling that a player is playing a game at the same time as anyone else on the planet requires a lot of effort, while developing multiplayer games. That is because it takes some time for the information to travel between two players. Therefore, there is one unique instance of the game created on every one gaming station. Thus, the key of creating a multiplayer game is the synchronization between these stations – the way that the player can not recognize it. The developer must make many decisions, that are based on the type of the game that is being developed. In this chapter, I will describe what possibilities the developer has. There will also be presented several techniques used while developing multiplayer games.

## 4.1 Turn based vs. Real-time games

Multiplayer games can be classed to many various genres. These genres can also be divided into two main types: turn based and real-time multiplayer games.

### Turn based games

As the name says, these are games, that are divided into turns. They assure that the game is divided into definite number of discrete steps, that lead to the end of the game. At one time, only one player can affect the game state. Every turn takes various length of time, that can be limited or unlimited – the player has time to make his turn. There are mostly 3 ending states, that can be reached by the player: victory, tie or failure. The typical turn based game is e.g. chess.

This kind of games was chronically first, mostly because of the simplicity of the implementation: there is no need to synchronize/update states of the game frequently and the amount of data is low. First online turn based games were played through mail. This kind of games is commonly used on all mobile platforms.

[1] Massively multiplayer online role-playing game
[2] real-time strategy
[3] first-person shooter

### Real-time multiplayer games

This type of games, on the other hand, is much more complicated. Although the game is a discrete simulation, the high frequency of game updates creates the feeling of immersion, reality. That is why the player has only limited time to think, react. At one moment multiple players can affect the game state. High frequency of game updates requires a good internet connection with large bandwidth and low latency. One big aspect related to real-time games is the fairness of the game. High latency and high amounts of lag can cause game inconstinency. Real-time games are also more proned to cheating than turn based games.

The usage of this kind of multiplayer games on Android and all other mobile platforms is still rising. Nowaday mobile devices such as smartphones and tablets provide a strong computational power and 3G mobile networks are still more capable of good quality connection with low latency.

## 4.2 Types of network models

Although there are many types of network models (or topologies), the two models described below are the most common and relevant to the topic of multiplayer games.

### 4.2.1 Peer-to-Peer model

Also called *distributed model*, is a network model, where all nodes are fully connected – figure 4.1. First multiplayer games like *Doom* or *Age of empires* [5] have been using this model, but today, it is not very common. The advantage of this model is mainly its simplicity. All clients are equal, communicate directly with each other and share their resources – for example bandwidth. They are also in full control of the game simulation on their station. If a client disconnects, and there are at least 2 clients communicating, the game can still run.

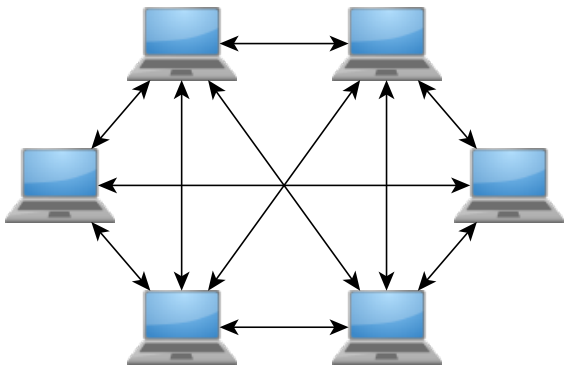


Figure 4.1: Peer-to-peer network model

However, if there is a client with a slow internet connection in the game, all clients must adjust the game speed (internal frame rate) and wait for its inputs. It is important, that the game must be deterministic on every station – every input from a player must be

processed equally on every client. Another downside of this model is the impossibility of joining an already started game. That is caused because the game state is not stored on any station (server) – it is changing on the ground of the inputs from other players. Also, with the growth of the number of connected clients the chance of network failure increases linearly.

### 4.2.2 Client-server model

Also called *centralized model*, represents a topology of stations, that are all connected to a single central node, the server – figure 4.2. Server receives all the user inputs, proceses them and then broadcasts them to all connected stations. It is responsible for all important decisions and helps to make the game state consistent [8]. This makes the server the most vulnerable part of this model. It needs to have high bandwidth and strong computing power to be able to process all the user inputs and run the code.

It is possible for a client to be also the server, but both parts are logically divided. In this models, clients in the network act just as dumb terminals, only sending changed inputs from the user. Nowadays, this model is being used in most of the multiplayer games, but with some modifications to it. Different approaches of the client-server model are discussed in section 4.3.

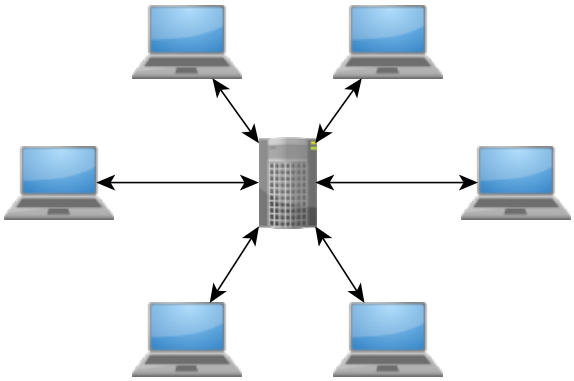


Figure 4.2: Client-server network model

## 4.3 Approaches of implementing a client-server communication for a real-time multiplayer game

As mentioned in 4.2.2, there are more approaches how to develop the server and client in the client-server model. These variations are described below.

### Non-authoritative server

This kind of a server simply receives user inputs from clients and broadcasts them to all connected clients. It lets the client to process the inputs from other players. That means, that the client is not just a dumb terminal, but it also runs code and is allowed to make decisions. The downside of this approach is vulnerability to cheating and high possibility of lag.

### Authoritative server

In this case, clients have only limited code. The whole game simulation runs on the server and only informations from the server are taken as valid. In FPS games like *Counter Strike* or *Half-Life*, the game is simulated on the server in discrete time steps called ticks [6]. All user inputs are being processed during these ticks, the server runs a simulation step and if necessary, after simulating a tick, the server sends a game update to the corresponding player.

Anyhow, this creates the fact, that on the client side the game does not look smooth. For example, if the player wants to move, he pushes a button and the client sends command to move. The server processes the command and after some time it sends back to the client the position of the player. So after pushing the button, it takes few moments (milliseconds) before the player starts to move - as shown on figure 4.3. To prevent this, clients are allowed to run some code to smooth the movement – with the *client prediction* technique.

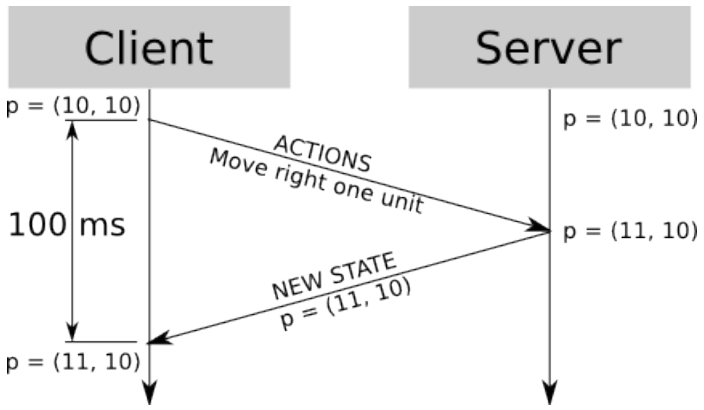



Figure 4.3: Effect of network delays [7]

### Techniques for hiding latency and low bandwidth issues

While playing a multiplayer game, players usually want to have a single player-like experience. Therefore, it is crucial to hide issues connected with latency and smooth the game as much as possible. To achieve this, there are some techniques, that are described in the paragraphs below.

**Client prediction** This technique is used to smooth the movement of the local player and prevent his jumping/shifting from one point to another. Before getting the new state of the player from the server, the client can calculate its next approximate state locally and initiate animation. After receiving the response from the server, the predicted position and the position from the server are compared [6]. If they differ, a prediction error has occurred. In this case, the client must correct player's position with the position of the authoritative server (that is why is the server called authoritative). This may cause an unpredictable shift to another position, so there is also need to use smoothing algorithms, to smooth the correction.

**Dead Reckoning** [1] This technique is used in the *Distributed Interactive Simulation* (DIS) protocol. It is used by the clients to simulate the motion of remote game objects locally. It uses old states of in-game entities, called *protocol data units* (PDU). This state contains informations about the entity, such as its position, velocity, orientation, acceleration, etc. These informations are used to extrapolate the next state of the object. However, some algorithms that are used for this calculation, need CPU with high computing power. Another downside of this technique – it is hard to be applied with objects/entities, that can unpredictably or quickly change their state (e.g. jump, turn around, etc.).

**Entity interpolation** In game genres like FPS it is impossible to predict next position of the game object. It is simply because in these games, players can change their state, velocity or direction instantly. Therefore, entity interpolation technique is used. It is another technique used to smooth the movement of remote game entities. The basis of this method is to render positions and animations of remote entities slightly in the past – in the position between two last acknowledged states from the authoritative server, which is interpolated from them (figure 4.4). This assures, that if the remote player makes a sudden change of his state, e.g. change his direction, there is still time to process it and the game runs still smoothly. But this also means, that the local player sees remote players in a small delay (in milliseconds) and the game is in a constant latency. But this small latency is not noticeable, even in first person shooters.

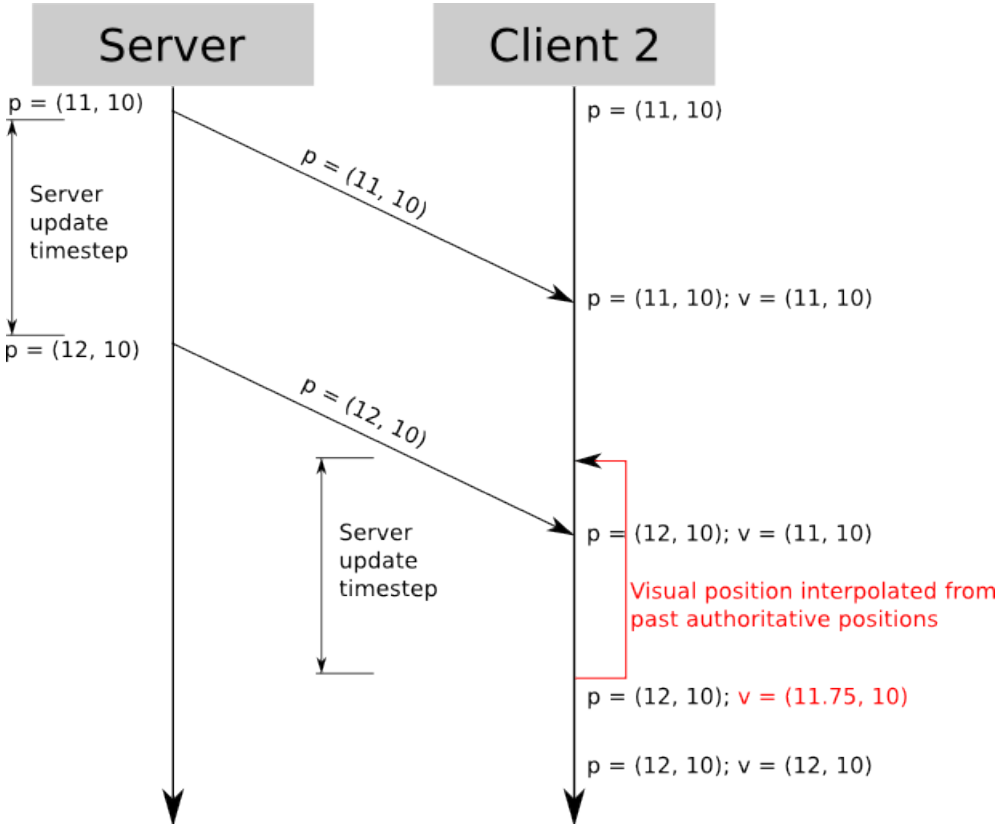



Figure 4.4: Client 2 renders Client 1 "in the past", interpolating last known positions [2]

**Delta compression** In multiplayer games, it is crucial to minimize the amount the data sent through the network. For example, if you update the game state 10 times per second, the network bandwidth can be consumed really quickly, when unnecessary data are being sent. That is why every networking model of the game should use delta compression. Delta compression is a technique, where the server sends to the client only data, that have changed since the last acknowledged state [9]. The key is to remember client's last state and compare it to the new one. An example could be player's movement. If the player is not moving, it is redundant to send the data about his position through the network.

# Chapter 5

# Third party network frameworks and game engines

This chapter discusses the usage of network and game frameworks, which may be used to create a network application or a game at Android platform. The first part is aimed at networking frameworks and their advantages and disadvantages. The next part describes the game engine used for the development of the game connected with this thesis. All links to framework websites are listed in appendix A.

### Benefits of using frameworks

Using a framework on any platform is a common practice, although it is not absolute necessity. It is up to the developer and the application that is being developed. Building an application with a lot of unnecessary and unused functions may be inappropriate. However, the main benefit of framework application is the **simplification of development and time saving**. By merging low-level or frequently used operations to practical methods, it allows the developer to focus on more important tasks. It is important "not to reinvent the wheel". The framework can extend the possibilities of functionality of the platform and add some features, that are missing. It can also help to improve the modularity and ease maintenance.

## 5.1 Network frameworks

In this section i will discuss all network frameworks from the internet, that I found and analyzed.

### Mages

This framework is no longer under development, however it is aimed for Android 2.2 version, and therefore, its compatibility with newer versions should be guaranteed (J2ME devices are also supported). It is written in Java and it supports features like lobby, creating games/joining games, chat messaging, game invitation, player ratings, etc.

However, it is aimed for turn-based board games or turn-based strategy and therefore it cannot be used in a real-time multiplayer game.

### Cubeia Firebase

A free, open source multiplayer project, that supports Java, Javascript, HTML5, Flash, C++, C# on the client side and Java, Ruby, Groovy and Python on the server side. It uses Maven – a building tool for Java projects. Cubeia does not provide a generated documentation (like Javadoc), but there is a wiki page with many tutorials for beginners. It can support real-time multiplayer games, like first person shooters, but the main trend of this framework are turn-based games (mainly card games like Poker). This fact and also user unfriendly Maven configuration files made me reject this solution.

### Skiller SDK

More than a network framework, Skiller is also a mobile social gaming platform, that supports Android, J2ME, Windows Phone 7 and Blackberry. It allows the developer to create multiplayer games (turn-based and real-time) and add social features like ranking, leaderboard, achievements, buddy lists, challenges, etc. There is also a possibility to monetize the game with advertising, creating virtual currency, in-app purchases and a billing system. Part of the system are some basic analytics tools. In order to use this framework, the developer has to register on the website and create an account (for free). The developer cannot create his own server locally, every client joins to Skiller servers. There is a documentation in Javadoc and also "getting started" page, but there are no examples for absolute beginners.

The impossibility to run the server locally and therefore editing it and creating server-side logic and also chaotic approach for the developers made this framework inappropriate for my application.

### Photon server

Another solution that supports almost all platforms for the client: Unity3D, Flash, Windows, Mac, Android, HTML5, iOS, etc. This framework is used for first-class commercial games and supports genres like MMORPG, FPS or racing games. Photon offers all features for real-time multiplayer games (e.g. joining/creating rooms, matchmaking, etc.), reliable UPD connection and high performance server, written in C/C++. The server itself can run only on Windows. It can be configured by user and server-side logic can be programmed with C# and Visual Studio. Free license allows the user to have a server with maximal 20 CCU[4]. Developers have access to many tutorials, examples and also an official forum.

Photon also offers Photon cloud. With this service the user does not need to run and configure a server locally, but he can join to one of the servers (USA, Europe, Korea, Japan, Singapore). However, it is not possible to have a custom server-side logic for the game.

This framework is a professional tool for developers working in commercial business. I decided not to use this framework for my application, because of pointless difficulty. Also, I wanted to develop both server and client on Mac OS, which is not impossible (running virtual machine), but again, needless.

## 5.1.1 SmartFoxServer

From all the network frameworks I found and analyzed on the internet, I decided to pick this one. In the paragraphs below I will list main features, that this platform provides and

[4]Concurrent users

also describe its basic functionality.

**General overview** This multiplayer platform, created by company gotoAndPlay(), offers three products: SmartFoxServer BASIC, PRO and 2X. Since the main development is focused on SmartFoxServer 2X, I decided to apply this version. These are the main features, that this framework offers:

- **Client API:** Flash (ActionScript 3), Unity (C#), iOS (Objective C), Android (Java, C++), HTML5 (JavaScript), .NET (C#), Java2 SE, Mac OS X, Windows 8
- **Server deployment on:** Windows, Linux (UNIX), Mac OS X
- **Security:** secure login mechanism, banning system, IP filtering, anti flood filter to prevent flooding attacks, permission manager for setting various user profiles with different permission options
- **BlueBox technology:** provides possibility for HTTP tunneling (for clients behind firewalls and proxies)
- matchmaking, game invitations, challenges, public/private games, chatting, lobby system ...

As listed before, the server can be deployed on all main platforms. It can be launched as a normal application, or as a service that runs on background. However, the developer has no access to edit its functionality, the server works like *black box*. The communication handling is hidden. If the developer wants to add some functionality to the server (like server-side logic), there is an option to use *Extensions*.

There is a lot of documentation material available for SmartFoxServer 2X. Besides of the generated API documentation for both client and server, there are many well-described example applications for each client platform, tutorials, video tutorials and also official forum.

There are more types of licensing, which differ by the number of allowed CCU. Free license allows the server to have maximum of 100 CCU, which is enough for my application.

### Events

The communication between clients and the server is driven by events. Events sent from the server should be handled by the client and vice versa. Each event contains informations needed to its handling (e.g. event "`USER_LOGIN`" contains object `User`, which represents the user, that has logged in).

### Zones and Rooms

Users, that are connecting to the server, come through sequential stages. However, this fact is hidden for the user (player). First, they connect to a Zone. Zone represents an isolated application, that runs in the SmartFoxServer, for example one Zone can be used for Poker game, second one for FPS, etc. After that, the user joins a Room. Rooms can represent a lobby room, chat room, or a game room – this depends on the application itself. Rooms can be merged into groups, as shown on figure 5.1. Users get only those event messages, that are relevant to the room group, that is joined.

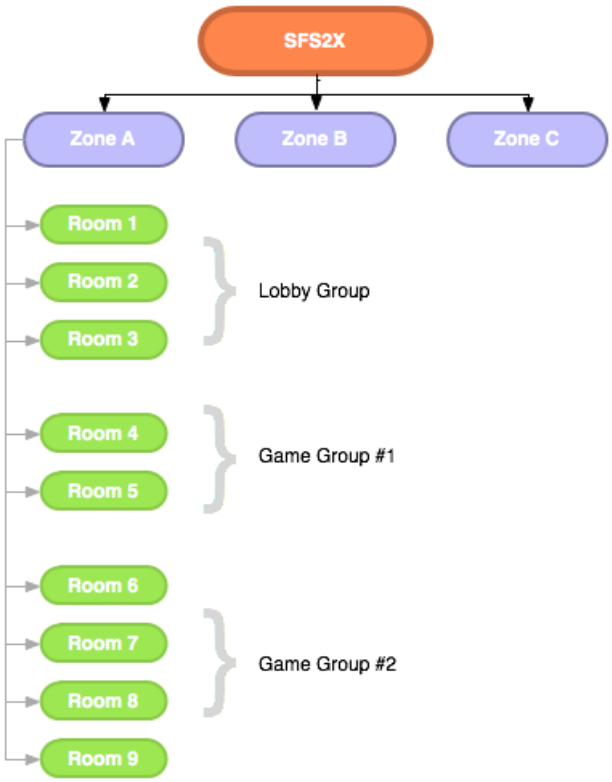



Figure 5.1: Rooms and Zones architecture in SmartFoxServer 2X [3]

### Extensions

In order to broaden the functionality of the client-server communication, the developer has to write Extensions. Each Extension is connected to one concrete Zone. All Rooms in this Zone contain one instance of the Extension (each Extension on separate thread). Because the core of the SmartFoxServer is written in Java, the Extension development is also targeted to this programming language. Extensions are deployed in a single .jar file, that is stored within the server.

### Administration tool

SmartFoxServer comes with Admin tool, which can be accessed through a browser. This tool allows the administrator to configure the server, check logs, traffic, memory and CPU consumption, kick or ban users, etc. It also allows the administrator to easily update SmartFoxServer to latest version.

## 5.2 Game engines

In this section the reader can find informations about game engines, that are available for the developers on Android platform. After that, there is a complex description about one concrete game framework, that was used for my bachelor thesis.

### Android game engines

Since the creation of Android, many commercial or community created game engines have been released. Alternativly, some frameworks, that have already been on the market, have extended their reach to this platform. In the table 5.1, there are listed some of the most used frameworks for game development.

| Name | 2D/3D | Platform(s) | License | Language |
|---|---|---|---|---|
| Unity3D | 3D | Cross-platform | Paid | C++ |
| Cocos2d-x | 2D | iOS, Android | Free | C++ |
| Proton SDK | 2D & 3D | Cross-platform | Free | C++ |
| jMonkeyEngine | 3D | Android | Free | Java |
| libGDX | 3D | Android, HTML5, Desktop | Free | Java |

Table 5.1: Table of widely used game frameworks

## 5.2.1 AndEngine

From all the game engines and frameworks, I found on the internet, I decided to pick this one. In this part I will describe main features, that this platform offers.

### General overview

AndEngine is one of the most popular game engines for Android. It is a 2D OpenGL platform created by Nicolas Gramlich. There are over 5000 applications developed with this framework.

There are two versions of AndEngine: GLES 1 and GLES 2. They are named after the versions of OpenGL ES (1.0 and 2.0), that they require the device to have. The development is focused on GLES 2 version, GLES 1 is no longer under development. However, if the developer wants support for older devices, he can use it as well. GLES 2 requires minimal Android version 2.2 (Froyo).

The support of this engine is driven by community. There is no official documentation (only generated Javadoc without comments), but, on the other hand, there are other sources available, which the developers can draw from. Firstly, there is the official AndEngine forum with a lot of articles, questions, tutorials, video tutorials and other materials. Also, many questions have been answered on stackoverflow.com. But the best usable material available are the official AndEngine examples, that show what this framework is capable of. Developers can download all the sources, test them and use them.

### Setup and extensions

The source code can by directly imported to Eclipse from the AndEngine GitHub repository. After that, the developer can create his project and add the AndEngine source code as a library project.

AndEngine offers officially more than 30 extensions, which can broaden options of development. The most used extensions are these:

- **Physics Box2D** – adds support for the popular Box2D physics engine

- **Collision Detection** – provides pixel-perfect collision detection
- **Live Wallpaper** – created AndEngine animation can be set as a wallpaper of the device
- **Augmented Reality** – enhances the use of camera and all sensors of the device
- **Multiplayer** – adds a simple client-server communication implementation based on their IP
- **SVG Texture Region** – allows .svg[5] formats to be used as a source of textures

### Main engine Entities

A Entity in AndEngine represents any object, that is a part of the simulated world – from lines, rectangles, ellipses, sprites, text to particles. All these object classes extend the Entity class. In the paragraphs below, main Andengine elements, relevant to the development of my application, are described.

**Camera** Camera represents a view of the user, that holds the device. Cameras can be:

- **Static** – The camera does not move. Its width and height parameters are set and never change.
- **Chasing** – The camera follows an Entity. For example, the camera can follow the movement of player's figure (however, camera boundaries should be set).

**Scene** Scene can be perceived as a container of all graphical elements, that are being rendered. It is a root element of Entity object hierarchy. Scene uses two-dimensional cartesian system, where the position [X,Y] = [0,0] is located on the top left corner of the display. A Scene may have a child scene. One of the subclasses of Scene is HUD[6] Scene. It is set to its parent scene and stays on the same position of the display. A HUD usually shows game informations for the player, e.g. score, health, minimap, etc.

**Sprites** A sprite can be defined as a 2D bitmap image, which is attached to a Scene on coordinates X and Y. There are multiple types of Sprites:

- **Sprite** – a single 2D bitmap image
- **TiledSprite** – a set of sprites from one image file – figure 5.2. Used mostly for buttons – its changed state can be loaded from a single file by defining the column and the row of the desired Sprite
- **AnimatedSprite** – a TiledSprite with added option of animation

Sprites can be loaded from .png, .jpg, .bmp or .svg files. They can be moved, rotated, skewed, scaled, resized; the programmer can change its color or opacity. There is also a possibility to attach a Sprite to another Sprite. All changes made to the child sprite will also apply to the parent.

[5] Scalable Vector Graphics
[6] Head-up display

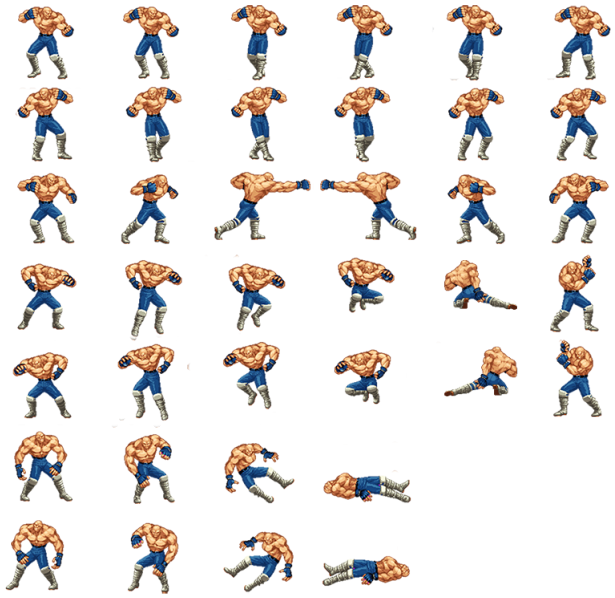


Figure 5.2: AndEngine TiledSprite

### Sprite animation

In AndEngine, it is possible to animate Sprites simply by changing their frames from the loaded tiled texture (figure 5.2). It is done by using Sprite's method `animate`. There are multiple parameters, that can be passed to this method:

- time duration of all/each each frame in milliseconds
- which frames are animated – e.g. {0-5}, {0,1,2,0}, etc. (numbering is shown on figure 5.3)
- how many times should the animation be looped (0 - infinity)
- `IAnimationListener` – it is possible to add a listener, which can listen to four events: `onAnimationStarted`, `onAnimationFinished`, `onAnimationLoopFinished` and `onAnimationFrameChanged`

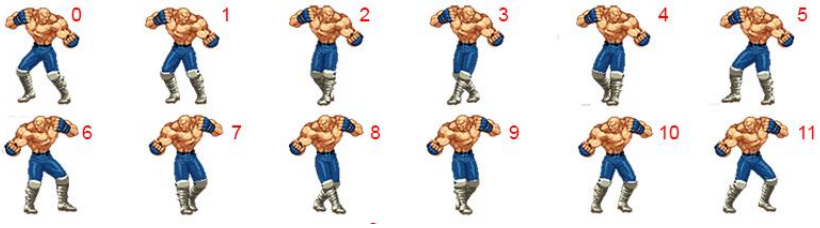



Figure 5.3: Numbering of frames used by animations of Sprites

### Entity Modifiers

In order to manipulate with Entities (move, scale, skew, change opacity, etc.) the developer can use Entity Modifier. Necessary arguments passed to this method is the time duration in seconds and the type of the modifier – effect (fade out, fade in, rotation ...) or path, that the entity will move with. The path may be created by hand (e.g. by cubic spline, an array of positions, etc.), however, AndEngine offers many pre-programmed move modifiers (like linear movement, accelerating movement, bouncing, etc.). The developer may also add an argument `IEntityModifierListener`, which listens to events `onModifierStarted` and `onModifierFinished`.

### Collision detection

To detect an intersection of two Entities on the Scene, AndEngine provides method `collidesWith(Entity)` for every Entity. However, the default collision detection in AndEngine may be unsatisfying for the developer, because it works as a bounding box check [4] – every shape is surrounded by axis-aligned rectangles and only the intersection between these rectangles is checked. This may cause a problem with objects with shapes, that are not convex. Fortunately, there is a Pixel Perfect Collision Detection Extension, which solves this problem – figure 5.4.

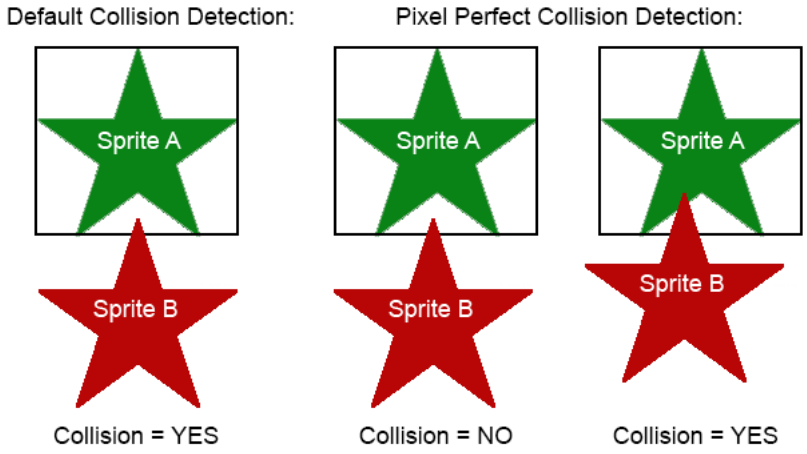



Figure 5.4: Default AndEngine Collision Detection vs. Pixel Perfect Collision Detection

# Chapter 6

# Application design

This chapter focuses on the design phase of game develoment. The reader will get to know the game concept and how the game is controlled. Afther that, design of the user interface is described.

## 6.1 Game concept

The concept of the game is quite straightforward. The created game is meant to be a real-time multiplayer 2D fighting game. The player can choose a character and fight against other 3 players in "death match" style – all players fight against each other and the last man standing wins the game. Characters can attack on short distance with punches/kicks and also use long range special attacks, that are constrained by the amount of their energy, that is refreshing over time. Besides moving and attacking, characters can also jump and use blocks, that can block any damage.

### Fighting

To achieve fairness and make the gameplay better and more fun, it is necessary to add some constrictions to fighting. For example, if there were no fighting constrictions, two players would meet and attack each other, untill one of them dies. Therefore, after the player makes 3 hits to another one, the stricken player falls to the ground. After few moments, he stands up and for a short amount of time, he cannot get any more damage. This way the attacking player has to move away a little bit and the fight can begin again. If the player gets hit, he is stunned for a moment and cannot attack back immediately.

If the player gets hit by the special attack, he receives more damage and falls to the ground immediately. Therefore, it is good to try blocking, or jumping over all special attacks.

### Application flow

The application can be divided into four parts: Main menu, lobby, game room and the game itself. At the application start, the user writes his nickname and connects to the lobby. Here, he can see all the players, that are connected to the lobby and the game rooms, that have been created. The user is able either to refresh the games list or to create a new game room. In the game room he can see other players, that are connected and what fighting character have they chosen. After character selection starts the game. When the game is

finished, the player is returned either to the game room or to the lobby. This process can be repeated all over again.

## 6.2 Game controls

Besides graphics, sound effects, the physics engine or the artificial intelligence, game controls have a significant influence on the game experience of the player. To ensure, that the game is fun and easy to play, the game controls must be intuitive and simple. In order to interact with the environment, the player can use various game controls. In this section, I will describe controls, that are used in the game.

### Virtual stick

For player movement, I decided to use virtual analog stick. Analog sticks have been used in history since the creation of first gaming platforms. All modern console platforms like PlayStation or Xbox use hardware controllers, that contain two analog sticks (e.g. in 3D games one for player movement and the other one for the camera rotation). This concept is also used on mobile platforms. A virtual analog stick, shown on figure 6.1, consists of two parts: the base and the knob. Using a finger (usually a thumb), the player touches the knob and moves it in any direction, in order to move his in-game character. The base of the virtual stick shows the boundary of the knob's position.

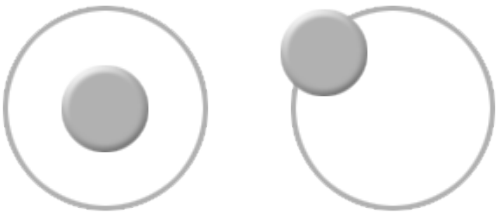


Figure 6.1: Virtual analog stick – with the default position of the knob and moved knob

### Swipe gestures

In order to attack, jump, block and use special attack, the player has to use swipe gestures – simple touch gestures, that react to the movement of the finger on the touchscreen. The finger can move in four directions: up, down, left and right. After the finger releases the touchscreen an action is invoked.

There is a possibility to use buttons for each of the actions. However, these buttons may take valuable place on the screen. On devices with smaller screens and low density, buttons take a lot of space and hide important places of the scene. Also, using four buttons for each action may be too much and chaotic. It may have a negative influence on the game experience.

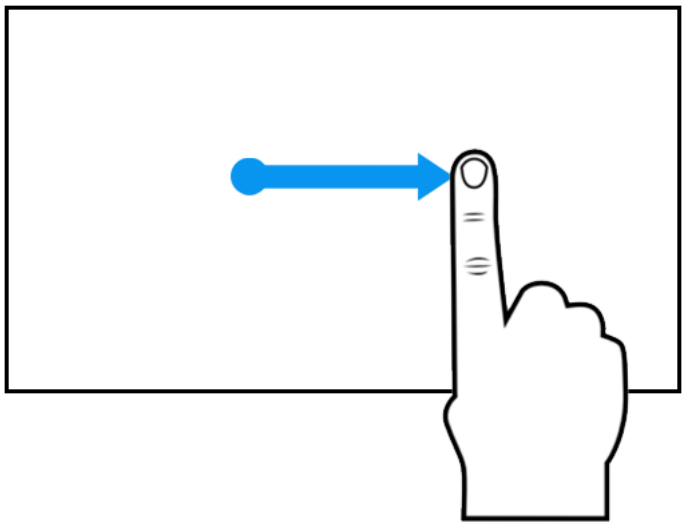


Figure 6.2: Simple left to right swipe gesture

## 6.3 User interface design

Focusing on the user interface (UI) is another important aspect of the application. User un-friendly interface may ruin whole experience while using the application and may lead to its extinction. In the paragraphs below there are figures with a short description about the main menu, lobby, game room and the game.

### Main menu

In main menu, the user can change his nickname in the text box, turn on/off sounds and music and tap the connection button to connect to the server. The nickname and sound preferences are remembered, so the next time the user launches the application, these informations are already set. The user interface of main menu is shown on figure 6.3.

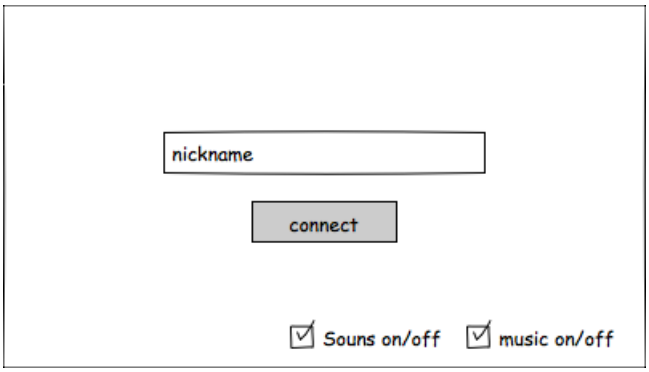



Figure 6.3: The user interface design of the main menu

### Lobby

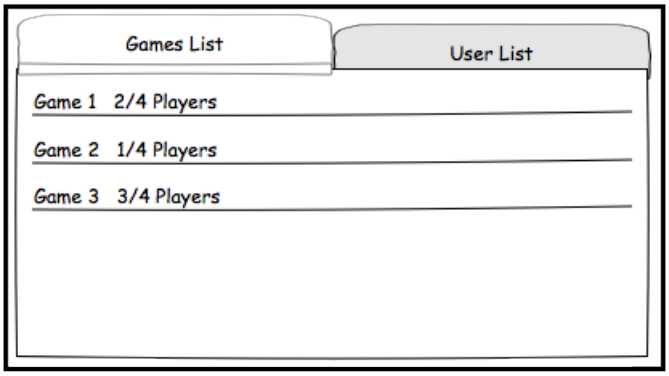



Figure 6.4: User interface design of lobby

The lobby consists of two lists – game list and user list. Each is situated in its own tab. Touching an item in the game list allows the user to connect to selected game room. After clicking the menu button of the Android device a pop-up menu appears. It offers the user to refresh the list, or create a new game. The user interface of lobby is shown on figures 6.4 and 6.5.

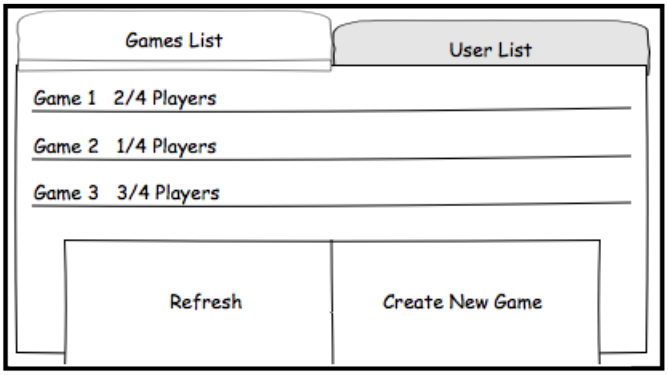



Figure 6.5: User interface design of lobby with a pop-up menu

### Game room

In the game room, it is necessary to show information about players, that have connected to it. Three informations are needed to be shown: player's nickname, player's chosen fighting character and a status, if the player is ready to play. By touching the image button, the player can choose his character. It is important to preserve the consistency of the game room, by prohibiting the player to touch (change) UI elements of other (remote) players. Design of the game room is shown on figure 6.6.

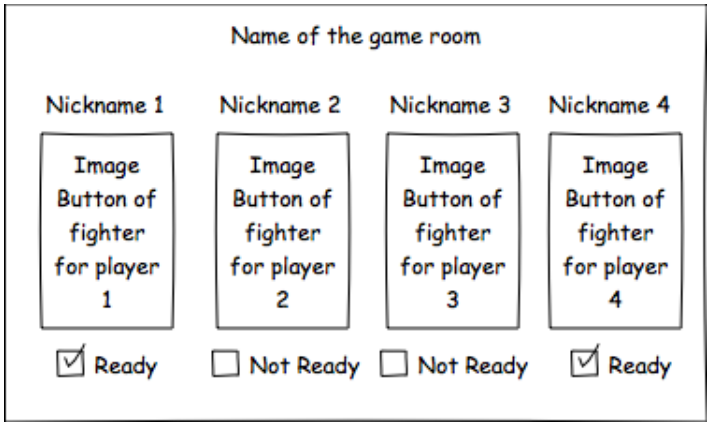



Figure 6.6: User interface design of the game room

### Game

The user interface of the game, shown on figure 6.7, is rather than simple. There is only one directly accessible UI element – a virtual analog stick located on the bottom left corner of the screen, used to move player's character. Under the figure of every player's character there are two bars: red bar representing player's health and blue bar representing player's energy. Players can move only on the fighting area, which is limited vertically (as shown on figure) and horizontally (borders of the display).

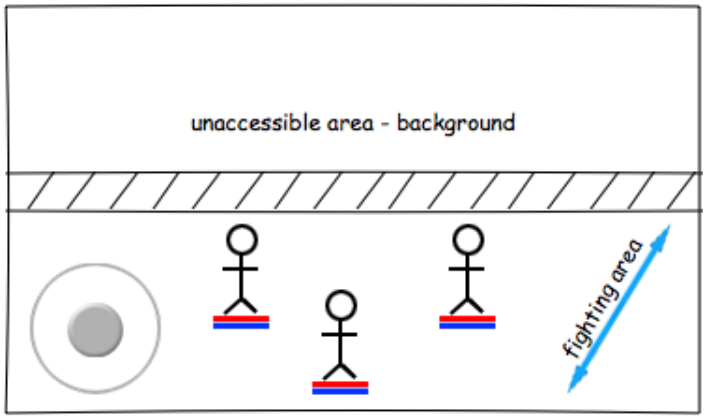



Figure 6.7: User interface design of the game

# Chapter 7

# Application implementation

The main goal of the implementation was to combine the game engine and network framework and make them work together. In this chapter, the reader will find out, how the application architecture looks like and what activities the application consists of. Later, there is a description of the network communication from separate views of the client and the server. Next part describes important parts of the game implementation and used tools, that helped while the development. Last part of this chapter is about the application testing.

### Application architecture

The application is divided in two parts: server and client. Server contains 15 classes, client 11. All classes in the client are grouped in one package. Server and client share one common class, `Player`, which is serialized. The structure of the client represents a typical Android project with default folders. The minimal target Android SDK version is 2.2 (Froyo). The AndEngine framework is added to the project as a project library, SmartFoxServer libraries are linked as .jar files.

### Activities transitions

There are 5 activities in the client application – figure 7.1. Splash Screen Activity is set to not be remembered in the activity back stack, so by hitting back button from Main Activity it is not possible to return to it. Activities Game Room and Game can exit with result code *RESULT_ERROR*, when there is something wrong. If the connection fails in the Game Room activity, the user is sent back to Main Activity. In the Game activity, if user disconnects, he is sent back to the Lobby.

## 7.1 Network communication

Main focus of the application is aimed at the network, i.e. messages, event handling, and the server-client communication in general. These topics are described in this section.

### Commands

In table 7.1, there is a list of commands, that are being sent between client and the server extension. The list contains informations about the command messages and data, that are sent with the message. Commands are sent trough UDP.

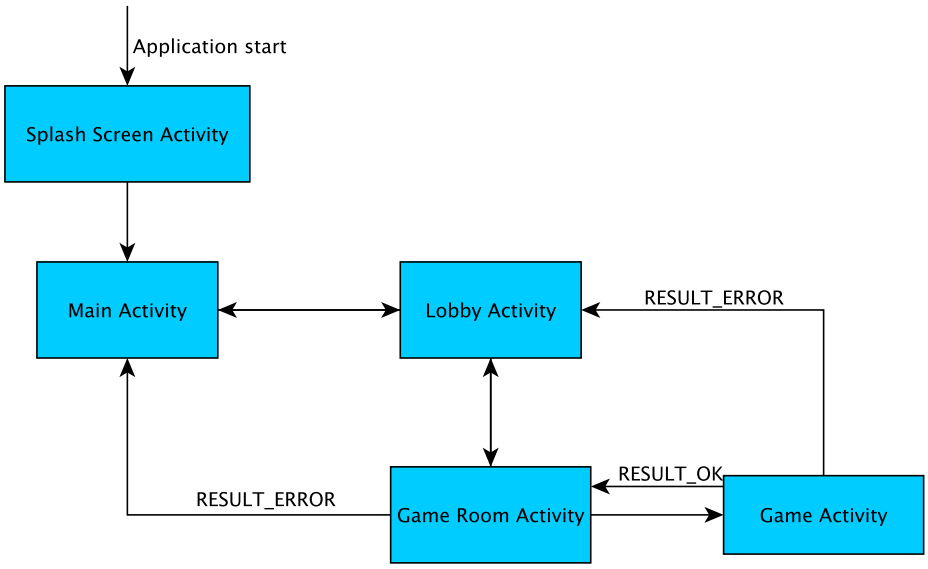



Figure 7.1: The flow of activities

| Command | Description | Data |
|---|---|---|
| start | sent when starting a game | null |
| refresh | sent to clear game data on the server | null |
| joinable | checks, if the room is joinable | `boolean` joinable |
| registerPlayer | sends player data to the server; if all data are send, server sends `start` message | class `Player` |
| move | player is moving | `int` player's ID, `float` position X, `float` position Y |
| attack | player attacked normally | `int` ID of the attacking player |
| specialAttack | player attacked with special attack | `int` player's ID |
| hit | player 1 attacked player 2 | 3 x `int` IDs of players, ID of attack |
| jump | player jumped | `int` player's ID |
| manDown | player got attacked 3 times and falls down | `int` player's ID |
| defend | player is blocking | `int` player's ID |
| energy | update player's energy | `int` player's ID, `double` energy value |

Table 7.1: Table of server extension commands

### 7.1.1 Server extension and configuration

The server uses default pre-installed Zone *BasicExamples* and one global Room *The Lobby*. In order to make the server extension work, all classes must be contained in one package named equally with client's package name. After exporting it to .jar file, it has to be stored

in the *extensions**name_of_the_extension* folder of the server folder hierarchy.

The extension contains class `bachelorThesisExtension.java`, which has defined, what class should be a message or event handled with. This class contains the list of players, that are currently playing. Most of the messages are handled simply by broadcasting the message to all players, except the one who sent it to the server. However, handling messages "*hit*" and "*specialHit*" is more complicated than that.

**Server-side hit confirmation** To ensure fairness and authority of the server, it is not possible to just accept the hit detection of the client. One player with high latency can see others in older positions, and therefore it is not possible to accept his hits, when other players are positioned somewhere else. Therefore, after player 1 hits player 2, two "*hit*" or "*specialHit*" messages must be sent – one says that the player 1 hits player 2 and the second one confirms the hit. After receiving this message the server first checks, if both players are close and therefore the hit is possible. Then it adds a record of the hit (class `attackRecord`) to a special map. This map of records consists of IDs of players 1 and 2 and the ID of the attack (normal/special). This way the server can see, if the hit was confirmed or not. After confirmation it sends message to the player, that was hit and his health has to be lowered.

### 7.1.2 Client-side communication

To connect to the server the client must create `SmartFox` object and use method `connect(server IP, port)`. This object represents the communication wit the server – this object sends all messages to the server. Therefore, it is static and it is used in all activities/classes.

Receiving messages from the server requires all activities to implement interface `IEventListener`, which requires to override method `dispatch`. This method has an argument `BaseEvent` – event (message) from the server. It consists of the event type and data. This interface is used in all activities, that communicate with the server (Lobby, Game Room, Game). Messages from the server extension have type `EXTENSION_RESPONSE` and contain the command (table 7.1) with necessary data.

**User Variables** With SmartFoxServer, it is possible to add special variables to any user. This is used in the Game Room Activity. Every connected player in the room is represented by object `Seat`. This object consists of player's nickname, an *Image Button* of his chosen character and a *checkBox*, that is checked, when the player is ready. After changing the *checkBox* status or character, player's user variable is changed. All clients receive message event of type `USER_VARIABLES_UPDATE`, which contains changed variables of the player.

## 7.2 Game Implementation

The game itself runs only on Game activity, which extends AndEngine's `SimpleBaseGameActivity`. This activity uses following classes:

- **Player** – represents a player. This class is serialized and can be sent directly to the server. It contains all necessary player informations (ID, health, energy, etc.).

- **GameManager** – consists of global variables needed in whole application, mathematical methods and methods for sending commands to the server extension. It also contains constants used in the game (e.g., animation times, movement speed, etc.). This way the game can be parameterized and therefore it is easier to change important values while adjusting the gameplay or testing.
- **Animator** – takes care of sprite animations. The movement animation also sets player's direction, which is used in other methods. While animating attacks, in it detects collisions with other players in concrete frames of the animation.

For debugging and testing purposes of client-side implementation parts, like animations or controls, Test Activity was created. It uses same methods and classes like Game activity, however, it does not use any communication methods.

### Used tools

The following tools helped me to develop the application in both design and implementation phases. All links of tools websites are listed in appendix A.

**BlueStacks App Player** This application allowed me to run Android applications on Windows and Mac. It supports GPU acceleration and therefore it is significantly faster than Android emulators. It is possible to adjust the screen size

**Pencil and yED** These tools were used in the application design phase. Pencil was used for creating user interface designs and prototypes. The second application, yEd, is a free diagram creator for Windows, Linux and Mac OS X.

**GIMP** This free image editing tool allowed me to create my own graphical elements like buttons, virtual analog stick or the splash screen. But it was mostly used while editing the sprite sheets.

## 7.3 Testing

The testing process can be divided into two phases. In the first phase, all the testing was done by myself. By using my smartphone (HTC Desire) and BlueStacks App Player, I managed to:

- tune in the network communication, to be as stable and fluid as possible
- adjust the user interface, to be more intuitive
- eliminate as many bugs as possible
- adjust the game parameters to enhance the gameplay

Because it is impossible to test a multiplayer game efficiently only with one person, in the phase two, more people have been involved to the testing. First, I explained them, how the application works. After that, they started to use the application, under my supervision. I observed, how they react to the user interface and their reactions during the gameplay. After the test, they were asked to fill an answer sheet with their feedback.

### Results of the testing survey

The application was tested on XY devices, including tablets. Multiple game scenarios were tested: game with 2, 3, 4 players, multiple created rooms with different numbers of players. Players were connected to each other on LAN only. The questions of the answer sheet and the results of the survey are shown on figure 7.2.

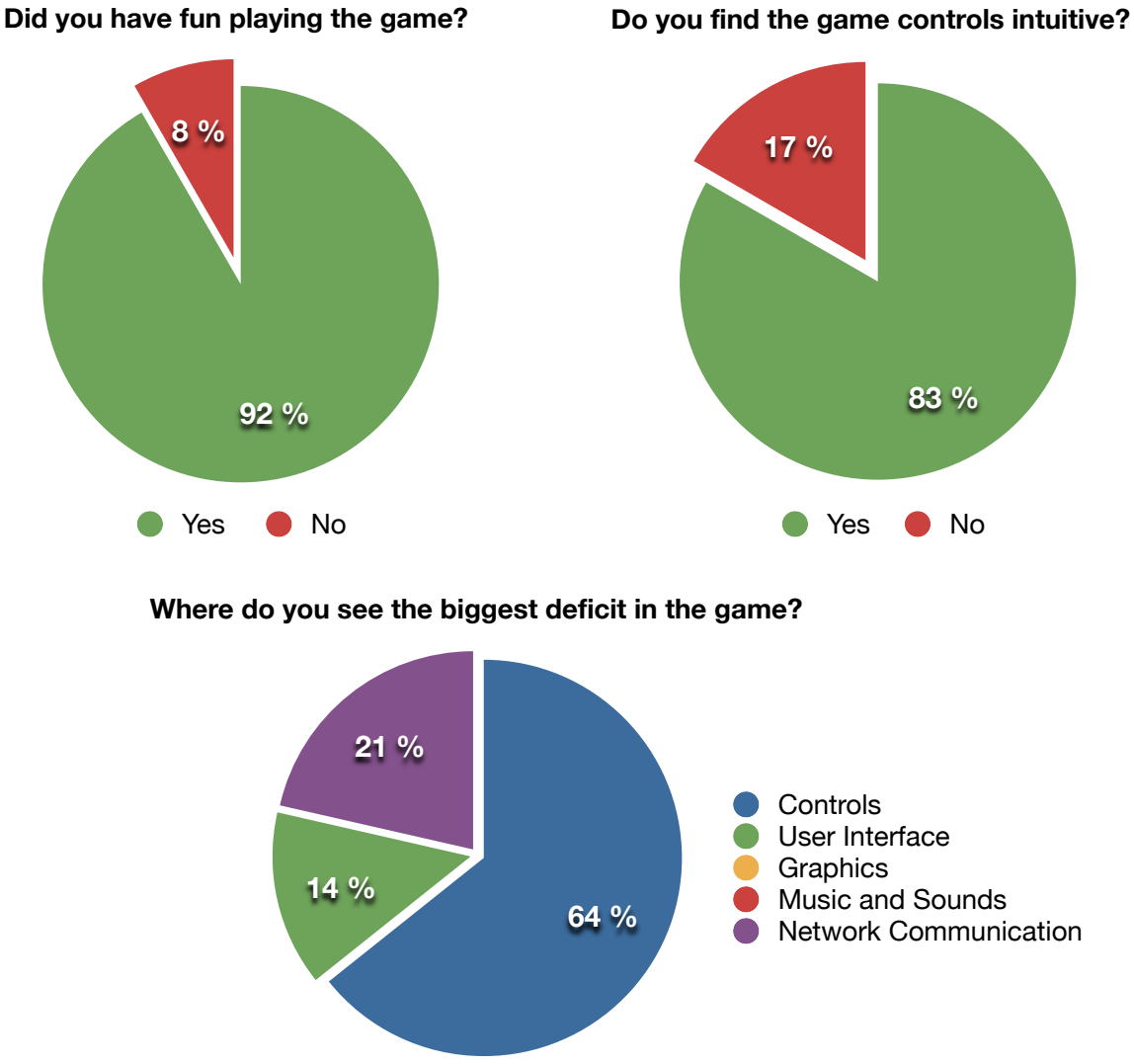



Figure 7.2: Survey results

# Chapter 8

# Conclusion

In this last chapter the reader can find the evaluation of the whole development process and the discussion about application improvements and its porting possibility to iOS platform.

## 8.1 Development process

The process of the work on the thesis was based on the steps described in its specification. First, I started to study the Android platform, its architecture and how to develop applications on it. After that, I began with the analysis of network and game frameworks on the internet. After choosing most suitable ones, I started with the design and implementation. In individual steps the lobby, game rooms and the game itself have been implemented. After making the network communication as stable and fluid as possible, I wanted to make the game fun to play. To ensure this, more players were needed. After gathering enough testing subjects and getting their feedback, some improvements in the user interface and gameplay settings were made.

## 8.2 Evaluation of the game and improvements for the future

The application successfully fulfilled the goal set in the specification of the bachelor thesis – I managed to find suitable network framework and game engine, combine them, make them work together and create a real-time multiplayer game. From the results of the survey (figure 7.2) I even managed to make the game fun to play. However, in order to make the experience with whole application better, there are many aspects, that have to be improved.

### Controls and user interface

The survey results showed, that the first thing, that should be improved, are game controls. Although testing subjects found it intuitive, swipe gestures did not suit all users. Solution to this problem may be to either change these gestures to some other controls (buttons, another virtual analog stick), or giving the user the possibility to choose from more options. However, this change goes hand in hand with the user interface of the game. Adding three or four buttons takes a lot of space on the display. In order to solve this, the game field could be moved up and on the bottom of the display, controls and HUD could be displayed.

User interface was created with only basic elements. It was adjusted to all types of displays. However, the UI was not focused enough. No animations, transitions were used.

In order to make the application nicer and therefore enhance the user experience, the user interface should be improved – by using a custom graphic design, adding transition animations, etc.

### Network communication

The connection quality could be evaluated as average. Most of the time, the communication appeared to run fluidly, with some occasional lags. However, on older or computationally weaker devices like HTC Desire or Samsung Galaxy Mini, the game did not run very well. It was caused with the inability of these devices to process many messages from the server (e.g. from other three players) at the same time. Newer devices like Sony Xperia U had no such problems and the game ran smoothly.

As mentioned before, the game was tested on LAN only. The average ping value was between 10 – 20 ms. It is quite possible, that if the game was tested on WAN or with a mobile network (3G,4G), the communication would not run fluidly, or work at all. The game was once tested on a slower router, with more than 10 devices connected – this caused, that the game was really chaotic (characters jumping from one position to another, sudden deaths, etc.) and therefore not really fun. In order to create a solution to these aspects, further tuning of the network would be needed – lowering the frequency of updates, focusing on better delta compression or implementing smoothing and lag compensation algorithms.

### Gameplay

The game was fun to play. However, it is clear, that it was fun only for once, or twice. To make the game fun repeatedly, more features are needed to be implemented. Firstly, more characters are necessary. Every character should be unique – with its own special attacks and abilities. Secondly, creating a combo attacking system. It is important to make the game a little bit harder to master, so that the player has to play it more frequently to gain skill in it. Also, adding more game types, like "team death-match", "capture the flag", or even creating a storyline would enhance the game experience.

Another aspect, that is not the most important, but surely very valuable, is the graphics. Creating scenes, that are nicely drawn with a parallax effect, shadows, or animations, can really immerse the player in the game. Also, adding better sound effects makes the game even more interesting.

## 8.3 Port to iOS platform

From the side of the network communication, the application is fully functional with almost any other platform. As mentioned in 5.1.1, SmartFoxServer provides support for iOS, Windows, Unity, Flash, Air or Java. Therefore, the network communication between my application and any of these platforms can be established and should work properly. All features used in my application work the same way on other platforms.

However, AndEngine is a not a cross-platform engine. Also, there is no equivalent engine for iOS or any other platform. In order to port my application to iOS, I would have to find a similar iOS game engine to AndEngine (e.g cocos2d) and implement all the classes and methods in its language. However, the logic and algorithms could remain all the same, so the development would not need to start from scratch.

# Bibliography


[1] Dead reckoning: Latency hiding for networked games. http://www.gamasutra.com/view/feature/3230/dead_reckoning_latency_hiding_for_.php, [cit. 2013-04-16]. [online].

[2] Fast-paced multiplayer (part III): entity interpolation. http://www.gabrielgambetta.com/wp-content/uploads/2010/12/cs3-02.png, [cit. 2013-04-16]. [online].

[3] Room architecture. http://docs2x.smartfoxserver.com/_documents/DevelopmentBasics/_images/rooms-and-groups.png, [cit. 2013-04-18]. [online].

[4] Basic collision detection in 2d – part 1. http://devmag.org.za/2009/04/13/basic-collision-detection-in-2d-part-1/, [cit. 2013-04-20]. [online].

[5] Microsoft Corporation. 1500 archers on a 28.8: Network programming in age of empires and beyond. http://zoo.cs.yale.edu/classes/cs538/readings/papers/terrano_1500arch.pdf, [cit. 2013-04-15]. [online].

[6] Valve Corporation. Source multiplayer networkng. https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking, [cit. 2013-04-15]. [online].

[7] Gabriel Gambetta. Fast-paced multiplayer (part II): client-side prediction and server reconciliation. http://www.gabrielgambetta.com/wp-content/uploads/2010/07/cs2-01.png, [cit. 2013-04-15]. [online].

[8] Brian Hook. Introduction to multiplayer game programming. http://trac.bookofhook.com/bookofhook/trac.cgi/wiki/IntroductionToMultiplayerGameProgramming, [cit. 2013-04-15]. [online].

[9] Brian Hook. The Quake3 networking model. http://trac.bookofhook.com/bookofhook/trac.cgi/wiki/Quake3Networking, [cit. 2013-04-16]. [online].

[10] Google Inc. Activity lifecycle. http://developer.android.com/images/activity_lifecycle.png, [cit. 2011-27-12]. [online].

[11] Google Inc. Android app framework. http://developer.android.com/about/versions/index.html, [cit. 2011-27-12]. [online].

[12] Google Inc. Android sdk. http://developer.android.com/sdk/index.html, [cit. 2011-27-12]. [online].

[13] Google Inc. Dashboards. http://developer.android.com/about/dashboards/index.html, [cit. 2011-27-12]. [online].

[14] Mark L. MURPHY. *Android 2: průvodce programováním mobilních aplikací*, volume 1. Brno: Computer Press, 2011. ISBN 9788025131947.

# Appendix A

# URL list

This appendix contains links all important websites, that are related to this thesis.

### Network frameworks

- Mages – http://code.google.com/p/mages
- Cubeia Firebase – http://www.cubeia.com/cubeia-firebase
- Skiller SDK – http://www.skiller-games.com
- Photon server – http://www.exitgames.com
- SmartFoxServer – http://www.smartfoxserver.com

### Game engines

- Unity3D – http://unity3d.com
- Cocos2d-x – http://www.cocos2d-x.org
- Proton SDK – http://www.rtsoft.com/wiki/doku.php?id=proton
- jMonkeyEngine – http://jmonkeyengine.org
- libGDX – http://libgdx.badlogicgames.com
- AndEngine – http://www.andengine.org

### Used tools

- BlueStacks App Player – http://www.bluestacks.com
- yED – http://www.yworks.com/en/products_yed_about.html
- Pencil – http://pencil.evolus.vn
- GIMP – http://www.gimp.org

### Media authors

- Character sprites – created by G.M.Spectre http://maximoff.alreadyread.net/SpriteSheets
- Flame effects – Pow Studios http://powstudios.com
- Music – created by ipkis.creativ, downloaded from http://www.jamendo.com
- Sound effects – all recorded by myself or downloaded from http://www.freesound.org